

A2
end

results. Numerous electronic technologies such as digital computers, calculators, audio devices, video equipment, and telephone systems have facilitated increased productivity and reduced costs in analyzing and communicating data, ideas and trends in most areas of business, science, education and entertainment. Electronic systems designed to provide these benefits often include integrated circuits on a single substrate that provide a variety advantages (e.g., such as miniaturization, fast processing of signals, minimal power consumption, etc.) over discrete component circuits. However, traditional approaches to manufacturing and designing single-chip systems are often very complex and consume significant resources.

Please replace the paragraph beginning at page 5, line 2 with the following:

A3

The present invention is a system and method of facilitating automatic generation of the boot sequence instructions in a convenient and efficient manner. In one embodiment of the present invention, an electronic device boot file generation method is utilized to create a boot file. A boot template file is created comprising special symbolic variable names that point to configuration registers within an electronic device. In one embodiment of the present invention the electronic device is a programmable device. User module selections are received with delineations of preferred configurations and functions associated with components of the programmable device. Application files are automatically generated based upon user module selections of configurations and functions utilizing the symbolic variable register names. Subsequent to user module selections, special symbolic variable register names are substituted or replaced with actual configuration register names. In one embodiment, a present invention programmable device boot file generation method also facilitates providing the appropriate interrupt vectors to their respective interrupt processing routines.

Please replace the paragraph beginning at page 5, line 19 with the following:

A4
In one exemplary implementation of the present invention an electronic device boot file generation method is performed by the electronic device's software design tool. The tool provides an interface for selecting applicable user modules, facilitates programming of desired functionality into the target device; and executes an assembler process. The tool produces a configuration image that defines configurations and functionalities of the electronic device and are stored in a memory included in the electronic device. The configuration image data may be generated and loaded on the electronic device in various manners including by one embodiment of a present invention. A software design tool comprising an editor for defining user module personalization and parameterization, an application editor for editing and compiling automatically generated and user created application program interface (API) code, and a debugger assisting debugging operations through emulation of the target device.

✓
Please replace the paragraph beginning at page 7, line 5 with the following:

A5
Figure 2A is a block diagram of one embodiment of a functional component depicted in greater detail.

✓
Please replace the paragraph beginning at page 7, line 9 with the following:

A6
Figure 3A is a flow chart of one embodiment of a present invention boot file creation method.

✓
Please replace the paragraph beginning at page 7, line 13 with the following:

A7
Figure 4 is a flow chart of a design tool process illustrating exemplary steps used by a design tool in accordance with one embodiment of the present invention.

✓
Please replace the paragraph beginning at page 9, line 21 with the following:

A8
Figure 1 is a block diagram showing a high level view of an exemplary integrated circuit (e.g., a microcontroller) 10 upon which embodiments of the present invention may be implemented. In one embodiment, integrated circuit 10 includes a communication bus 11, static random access memory (SRAM) 12, central processing unit (CPU) 14, flash read-only memory (ROM) 15, input/output (I/O) pin(s) 18 and functional component 25. Communication bus 11 is electrically coupled to static random access memory (SRAM) 12, central processing unit (CPU) 14, flash read-only memory (ROM) 15, input/output (I/O) pin(s) 18 and functional component 25. Static random access memory (SRAM) 12 stores volatile or temporary data during firmware execution. Central processing unit (CPU) 14 processes information and instructions. Flash read-only memory (ROM) 15 stores information and instructions (e.g., firmware). In one embodiment of the present invention, flash read-only memory (ROM) 15 stores configuration image data. Input/output (I/O) pin(s) 18 provides an interface with external devices (not shown). Functional component 25 is programmable to provide different functions and configurations.

Please replace the paragraph beginning at page 10, line 14 with the following:

A9
It is appreciated that integrated circuit 10 is readily adaptable to include a variety of other components. In one exemplary implementation, integrated circuit 10 also includes a dedicated functionality internal peripheral component 17 which is coupled to system bus 11 in addition to the functional component 25. An optional test interface (TI) may be coupled to integrated circuit 10 via a test interface coupler (TIC), which may be detachable, to perform debugging operations during startup and initialization of the integrated circuit. In one embodiment of the present invention, additional functions such as clocking and power control are provided by a variety of components including a precision oscillator and phase locked loop (PLL), a voltage reference, a 32 kHz crystal oscillator (which may be utilized for a variety of applications such as calibration and synchronization, etc.), an interrupt controller (for generating interrupt signals as required), a power

A9
end
on reset control unit (for performing functions related to power supply stability), and a brown-out detection unit (which detects substandard, subnominal power system parameters).

Please replace the paragraph beginning at page 11, line 8 with the following:

A10
Referring to Figure 2A, an embodiment of functional component 25 is depicted in greater detail. In this embodiment, functional component 25 includes an analog functional block 230, a digital functional block 240, and a programmable interconnect 250. In one exemplary implementation, analog functional block 230 includes a matrix of interconnected analog functional blocks A1 through AN. The number N may be any number of analog functional blocks. Likewise, digital block 240 includes a matrix of interconnected digital functional blocks D1 through DM. The number M may be any number of digital functional blocks.

Please replace the paragraph beginning at page 12, line 14 with the following:

A11
The present invention is readily adaptable for use with numerous functional blocks that are programmably configurable to provide a variety of functions. Exemplary functional peripherals include timers, controllers, serial communications units, Cycle Redundancy Check (CRC) generators, Universal Asynchronous Receiver/Transmitters (UARTs), amplifiers, programmable gain components, digital to analog converters, analog to digital converters, analog drivers, and various filters (e.g., high-, low-, and band-pass). In one exemplary implementation higher order user modules (e.g., modems, complex motor control, sensor devices, etc.) are created with combinations of functional blocks. Co-pending commonly-owned incorporated U.S. Patent Application Serial No. 10/033,027 (attorney docket number CYPR-CD00232), filed October 22, 2001, entitled "PROGRAMMABLE MICROCONTROLLER ARCHITECTURE," which is hereby incorporated by this reference, includes additional details on exemplary implementations of present invention

A11
end
integrated circuits (e.g., integrated circuit 10) and functional components (e.g., functional component 25).

✓
Please replace the paragraph beginning at page 13, line 20 with the following:

A12
In one embodiment of the present invention, a functional component (e.g., functional component 25) includes registers that are programmably configurable to store configuration data that defines the combination (e.g., electrical coupling) of the functional blocks and the characteristics (e.g., parameters) of the respective functional block elements. When a value is changed in a configuration register, the configuration and/or functionality of a corresponding integrated system 10 component is changed accordingly. In one exemplary implementation of the present invention, some functional blocks are configured to affect autonomous system operations, such as interrupts.

✓
Please replace the paragraph beginning at page 15, line 18 with the following:

A13
In the present exemplary embodiment, programmable interconnect 250 comprises a configuration system and a global mapping system. The configuration system is coupled to communication bus 11 and the global mapping system, which in turn is coupled to functional component 25. The configuration system is programmably configurable to selectively couple with communication bus 25 and/or the global mapping system. The global mapping system facilitates selective coupling of functional blocks included in functional component 25 to other functional blocks and/or communication bus 11. In one exemplary implementation, the global mapping system includes an input global mapping component and an output global mapping component.

✓
Please replace the paragraph beginning at page 17, line 12 with the following:

A14
In one embodiment of the present invention, a system timing block is included to provide timing information used for synchronizing and otherwise effectuating interfacing between system

AM end
functionalities (e.g., functional blocks). The system timing block like the functional component 25 is programmable. Advantageously, this allows the system timing block to generate a myriad of different time bases, as required for any particular application the system is being configured to effectuate. These time bases may be fed into analog functional blocks and digital functional blocks for use therein, via the programmable interconnect. Examples of analog functions requiring such time bases include conversions, modulations, and the like. One striking example of a digital function requiring such time bases is a universal asynchronous receiver transmitter (UART) functionality.

✓
Please replace the paragraph beginning at page 18, line 3 with the following:

AL5
There are a variety of critical system files that control the configuration of a programmable electronic device. In one exemplary implementation of the present invention, a boot file (e.g. boot.asm) that defines the boot sequence and controls boot operations resides in a source tree (e.g., under Source Files). In one embodiment of the present invention, the tool providing the automatic creation of assembly source code files (e.g., based upon user module descriptions) utilizes a boot template to generate the boot source code included in the boot file. In one exemplary implementation of the present invention, the template file comprises special symbolic variable register names that stand for (or point to) configuration registers. In one embodiment, the template file also facilitates mapping of appropriate interrupt vectors to their respective interrupt processing routines.

✓
Please replace the paragraph beginning at page 18, line 15 with the following:

AL6
When design changes are made to configuration and functions of the components and configuration image source code is automatically generated it is very difficult to track the actual configuration registers that are assigned to a user module, the appropriate values for loading into configuration registers, and respective appropriate interrupt vectors for interrupt processing routines.

Alle
end

It is significantly easier to track appropriate values with special symbolic variable register names during changes and automatic generation of configuration image code. After a configuration image is automatically generated, symbolic substitution is then performed to substitute the actual register name with its variable symbolic name place holder, and to assign appropriate interrupt vectors to interrupt processing routines. The template file facilitates substitution operations by providing a correlation between the special symbolic variable register names and the actual register names. The template file facilitates substitution operations by providing a correlation between appropriate interrupt vectors and their correct respective interrupt processing routines.

✓
Please replace the paragraph beginning at page 19, line 18 with the following:

AIM

Figure 3A is a flow chart of one embodiment of a present invention boot file creation method 300. Boot file creation method 300 facilitates creation of a boot file. In one embodiment, boot file creation method 300 utilizes symbolic register names when changes are being made to a configuration or boot file. This relieves a user from tedious manual tracking of actual configuration registers and complicated impacts changes have on the assignment of a particular configuration register or interrupt vector for a particular user module.

✓
Please replace the paragraph beginning at page 20, line 5 with the following:

A18

In one embodiment of the present invention, boot file creation method 300 is implemented on a design tool (e.g., a computer implemented software design tool). Additional details on an exemplary implementation of a present invention design tool are set forth in co-pending commonly-owned United States Patent Application Serial No. 09/989,570, filed November 19, 2001 (attorney docket number CYPR-CD01167M), entitled "METHOD FOR FACILITATING MICROCONTROLLER PROGRAMMING", which is hereby incorporated by this reference.

Please replace the paragraph beginning at page 20, line 17 with the following:

1A
In step 310, variable symbolic register names are assigned to a user module. In one embodiment of the present invention, a user module is a preconfigured function that may be based on more than one functional block.

Please replace the paragraph beginning at page 21, line 1 with the following:

A20
In step 320 an association is established between the variable symbolic register names and actual configuration register names. In one embodiment of the present invention, a boot template is utilized to establish the association between the variable symbolic register names and actual configuration register names. In one embodiment of the present invention, an association is also established between appropriate interrupt vectors and interrupt services routines.

Please replace the paragraph beginning at page 21, line 8 with the following:

A21
In step 330 a symbolic substitution is performed to replace the variable symbolic register names with actual configuration register names. In one embodiment of the present invention, the boot template is utilized to perform the symbolic substitution. The symbolic substitution replaces the variable symbolic register names with the actual configuration register names. In one embodiment of the present invention, the symbolic substitution includes automatic interrupt vector mapping that assigns the appropriate interrupt vector to an interrupt service routine. Figure 3B is a table illustrating one embodiment of an interrupt vector mapping table of the present invention. In one exemplary implementation of the present invention, the symbolic symbols located at specific ROM locations are replaced with actual instructions so that the interrupt vector picks up the correct instruction for a user module.

Please replace the paragraph beginning at page 22, line 1 with the following:

A22 In step 340 the boot file is loaded on a target device. In one embodiment the boot file is created and loaded on a target device by a design tool.

✓
Please replace the paragraph beginning at page 22, line 4 with the following:

A23 In one embodiment of the present invention, configuration images are provided by a design tool. Figure 4 is a flow chart of a design tool process 400, which illustrates exemplary steps used by a design tool in accordance with one embodiment of the present invention. One embodiment of an exemplary computer system utilized to implement design tool process 400 is set forth in incorporated United States Patent Application Serial No. 09/989,570, filed November 19, 2001 (attorney docket number CYPR-CD01167M), entitled "METHOD FOR FACILITATING MICROCONTROLLER PROGRAMMING". Design tool process 400 facilitates configuration, programming, building, emulation and debugging of a customized "target device." In one exemplary implementation the target device is similar to integrated circuit 10 of Figure 1 with a functional component 25 similar to Figure 2A. In one embodiment, design tool process 400 is carried out by a computer system under the control of computer-readable and computer-executable instructions. The computer-readable and computer-executable instructions reside, for example, in data storage features of the computer system such as a computer usable volatile memory, computer-usable non-volatile memory and/or data storage device. The computer-readable and computer-executable instructions direct computer system operations in accordance with design tool process 400.

✓
Please replace the paragraph beginning at page 23, line 3 with the following:

A24 In step 410, an interface for selecting applicable "user modules" is provided. In one embodiment of the present invention, a user module is a preconfigured function that may be based on more than one functional block. In one exemplary implementation a user module when programmed and loaded on a memory of the target device directs a functional block to work as a

A24
peripheral on the target device. At any time in design tool process 400, user modules may be added to or removed from the target device. The selected user modules are associated with (e.g., “placed” or “mapped to”) representations of functional blocks defined in the design tool. Once a user module is associated with a representation, its parameters can be viewed and modified as needed. Global parameters used by a plurality of user modules (for example, CPU clock speed) can also be set. In one embodiment of the present invention, interconnections between selected user modules are specified (e.g., either as each user module is placed or afterwards). The pin-out for each block can also be delineated, making a connection between the software configuration and the hardware of the target device.

✓
Please replace the paragraph beginning at page 24, line 3 with the following:

A25
In step 430, programming of the desired functionality into the target device is facilitated. In one embodiment of the present invention, source code files can be edited, added or removed. In one embodiment of the present invention, programmable configuration of external device ports is also facilitated by design tool process 400.

✓
Please replace the paragraph beginning at page 24, line 13 with the following:

A26
In step 450 the target device is “built” within the design tool. Building the target device in the design tool includes linking the programmed functionalities of the source files (including device configuration). In one exemplary implementation of the present invention, the linked programmed functionalities and the source files are downloaded to an emulator for debugging in step 450.

✓
Please replace the paragraph beginning at page 25, line 5 with the following:

A27
In step 470 a configuration image generated using design tool process 400 is loaded into memory of a target device. In one embodiment of the present invention a plurality of configuration images are loaded into memory of a target device.

✓
Please replace the paragraph beginning at page 25, line 9 with the following:

A28
Although specific steps are disclosed in design tool process 400 of Figure 4, such steps are exemplary. That is, the present invention is well suited to use with various other steps or variations of the steps recited in process 400. Additionally, for purposes of clarity and brevity, the discussion is directed at times to specific examples. The present invention design tool process 400, however, is not limited solely to use to design a particular target device (e.g., a microcontroller). Instead, the present invention is well suited to use with other types of computer-aided hardware and software design systems in which it is necessary to accomplish a multitude of tasks as part of an overall process.

✓
Please replace the paragraph beginning at page 25, line 19 with the following:

A29
In one embodiment of the present invention the components of a target device follow a boot sequence. Figure 5 is a flow chart of one embodiment of a present invention boot sequence. The boot sequence is dictated by the information included in a boot file (e.g., boot.asm).

✓
Please replace the paragraph beginning at page 26, line 3 with the following:

A30
In step 510, a reset vector for the source code is originated after a target device is powered up.

✓
Please replace the paragraph beginning at page 26, line 5 with the following:

A31
In step 520, an interrupt table is held for code that is executed when an interrupt occurs. In one embodiment of the present invention, the interrupt table is held while a symbolic substitution process (e.g., step 330 of boot file creation method 300) is utilized to provide the appropriate interrupt service routine code. In one exemplary implementation, a boot table (e.g., boot.tbl) is used for automatic generation of interrupt vector table entries that map a user module interrupt to appropriate interrupt service routine code in a correct ROM location. Correct device configuration initialization successfully occurs because an appropriate substitution file that provides correct values and interrupt service routine code in the right location is created (e.g., with a call to LoadConfig) for each configuration change.

✓
Please replace the paragraph beginning at page 26, line 16 with the following:

A32
In one embodiment of the present invention, some characteristics of the interrupt table are determined by the hardware of the target device. In one exemplary implementation, the hardware dictates that the interrupt entry points are 4 bytes apart beginning at program counter location zero (e.g., reset). Local jumps are enforced when an interrupt handler is originated (ORGed) within close proximity to (e.g., exactly at) the interrupt vectors. Therefore, a "long jump" (e.g., 3 byte long call) may be utilized and the vector table will not fall out of alignment (will not overrun into the next interrupt vector instruction).

✓
Please replace the paragraph beginning at page 27, line 5 with the following:

A33
The present invention conveniently provides automatic interrupt routing based upon the selection and configuration of user modules. One example of correct automatic interrupt routing includes the exemplary long jump discussed above. A user module that requires 4 bytes of a different kind of code that is extremely fast (e.g., increment a number and then return from the interrupt) is another example of an automatic interrupt service routine code generation that can be

A33
end

performed by the present invention. For example, a present invention design tool has the ability to occupy the 4 bytes in an optimal manner for an interrupt. If the user module is moved (e.g., 5 blocks over) appropriate interrupt service routine code will automatically be generated by a present invention symbolic substitution process (e.g., boot file creation method 300) relieving the user from manually tracking and providing this information.

✓
Please replace the paragraph beginning at page 28, line 7 with the following:

A34

The boot file is regenerated every time device configurations change and application files are generated. In one embodiment of the present invention, this ensures that interrupt handles are consistent with the configuration. Whenever the application files change (e.g., due to changes in the configurations or functions of a device design) the boot.asm is regenerated to ensure that interrupt handlers are consistent with the configuration. In one embodiment of the present invention the template file for the boot file (e.g., boot.tpl template for boot.asm) belongs to the design tool. It is recommended that a user does not make changes to the boot.asm file. However, in one embodiment of the present invention a user is permitted to hard code a change in boot.tpl file if a user does not want changes to a boot.asm file overwritten.

✓
Please replace the paragraph beginning at page 28, line 19 with the following:

A35

During the automatic source code generation processes of the present invention, the boot.asm file is constructed using the boot.tpl. For example, symbolic register names are replaced with the actual register names and appropriate interrupt vectors are mapped to their correct respective interrupt processing routines. The final boot.asm file is then completed. It is appreciated that the present invention substitution process is readily adaptable to a variety of applications, for example other source code files can also be automatically created using the present invention template